Amortized Analysis

- In amortized analysis, we are looking for the time complexity of a sequence of n operations, instead of the cost of a single operation.
- Cost of a sequence of n operations = n S(n), where S(n) = worst case cost of each of the n operations
- ***Amortized Cost*** = T(n)/n, where T(n) = worst case total cost of the n operations in the sequence.
- Amortized cost can be small even when some operations in that sequence are expensive. Often, the worst case may not occur in every operation. The cost of expensive operations may be 'paid for' by charging to other less expensive operations.

Problem 1: Stack Operations

- Data Structure: **Stack**
- Operations:
  - *Push(s,x)* : Push object *x* into stack *s*.
    - Cost: T(push)= O(1).
  - *Pop(s) :* Pop the top object in stack *s*.
    - Cost: T(pop)=O(1).
  - *MultiPop(s,k) ;* Pop the top *k* objects in stack *s*.
    - Cost: T(mp) = O(size(s)) worst case
- ***Assumption:*** Start with an empty stack
- ***Simple analysis:*** For N operations, the maximum size of stack is N. Since the cost of *MultiPop* under the worst case is O(N), which is the largest in all three operations, the total cost of N operations must be less than N x T(mp) = $O(N^2)$.

Amortized analysis: Stack Operations

- **Intuition**: Worst case cannot happen all the time!
- **Idea**: pay a dollar for every operation, and then count carefully.
- Suppose we pay 2 dollars for each *Push* operation, one to pay for the operation itself, and another for "future use" (we pin it to the object on the stack).
- When we do *Pop* or *MultiPop* operations to pop objects, instead of paying from our pocket, we pay the operations with the extra dollar pinned to the objects that are being popped.
- So the total cost of N operations must be less than 2 x N
- **Amortized cost** = T(N)/N = 2.

Problem 2: Binary Counter

- Data Structure: binary counter b.
- Operations: Inc(b).
  - Cost of Inc(b) = number of bits flipped in the operation.
- What's the total cost of N operations when this counter counts up to integer N?
- ***Approach 1: simple analysis***
  - The size of the counter is log(N). The worst case will be that every bit is flipped in an operation, so for N operations, the total cost under the worst case is O(Nlog(N))

Approach 2: Binary Counter

- **Intuition**: Worst case cannot happen all the time!

000000
000001
000010
000011
000100
000101
000110
000111

Bit 0 flips every time, bit 1 flips every other time, bit 2 flips every fourth time, etc. We can conclude that for bit k, it flips every 2^k time.

So the total bits flipped in N operations, when the counter counts from 1 to N, will be = ?

$$T(N) = \sum_{k=0}^{\log N} \frac{N}{2^k} < N \sum_{k=0}^{\infty} \frac{1}{2^k} = 2N$$

So the amortized cost will be T(N)/N = 2.

Approach 3: Binary Counter

- For k bit counters, the total cost is

  $t(k) = 2 \times t(k-1) + 1$
- So for N operations, $T(N) = t(\log(N))$.
- $t(k) = ?$
- $T(N)$ can be proved to be bounded by $2N$.

Amortized Analysis: Potential Method

- For the n operations, the data structure goes through states: D_0, D_1, D_2, …, D_n with costs c_1, c_2, …, c_n
- Define potential function $\Phi(D_i)$: represents the potential energy of data structure after i_{th} operation.
- The amortized cost of the i_{th} operation is defined by:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

- The total amortized cost is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{N} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} c_i$$

$$\sum_{i=1}^{n} c_i = -(\Phi(D_n) - \Phi(D_0)) + \sum_{i=1}^{n} \hat{c}_i$$

Potential Method - Cont'd

- If $\Phi(D_n) \geq \Phi(D_0)$

  then $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$

  which then acts as an upper bound for the total cost.

  So we need to define a suitable potential function such that this function is always **non-negative**.

Potential Method: Stack

- Define $\Phi(D)$ = # of items on stack
- $\Phi(D_0) = 0$
- $\Phi(D_n) \geq 0$

$$\hat{c}_{push} = c_{push} + 1 = 2$$

$$\hat{c}_{pop} = c_{pop} - 1 = 0$$

$$\hat{c}_{multipop(k)} = c_{multipop(k)} - k = k - k = 0$$

$$\sum^{N} c < \sum^{N} \hat{c} = \sum \hat{c}_{push} + \sum \hat{c}_{multipop} + \sum \hat{c}_{pop} = \sum \hat{c}_{push} < 2N$$

Potential Method: Binary Counter

- Define $\Phi(D)$ = # of 1's in counter
- $\Phi(D_0) = 0$
- $\Phi(D_n) \geq 0$

$$\hat{c} = c + \Delta\Phi = (k+1) + (1-k) = 2$$

$$\sum^{N} c < \sum^{N} \hat{c} = 2N$$